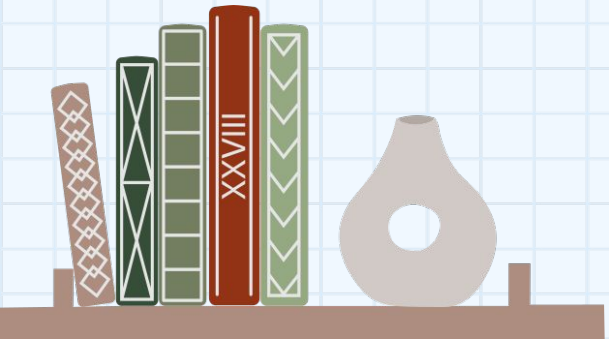
Polynomial Evaluation in Python

Understanding Basic Algorithms



Introduction

What is a Polynomial?

A polynomial is a math expression with variables and coefficients combined using addition, subtraction, and multiplication with non-negative integer exponents.

Degree and Coefficients:

- Degree: The highest exponent in the polynomial.
- Coefficients: The numerical factors of terms.

Examples:

- Linear: $P(x) = 2x + 3$
- Quadratic: $P(x) = x^2 - 4x + 7$
- Cubic: $P(x) = 4x^3 + 2x^2 - x + 5$

Why To Evaluate Polynomials?

We evaluate a polynomial to determine its value for a given input **x**. This has applications in:

- **Mathematics** (solving equations)
- **Physics** (motion equations)
- **Computer Science** (graphics, data fitting, AI)

Example: Evaluating a polynomial manually

$$P(x) = 3x^3 + 2x^2 - 5x + 7$$

For x = 2,

$$P(2) = 3(2)^3 + 2(2)^2 - 5(2) + 7 = 24 + 8 - 10 + 7 = 29$$

❖ No Python code needed here as it's just a mathematical evaluation.

Polynomial Representation in Python

```python
x = 2
P_x = 3 * (x**3) + 2 * (x**2) - 5 * x + 7
print(P_x)  # Output: 29
```

```
29
```

How to Store a Polynomial in Python?

- **Using a List**

  Standard Form Representation: Terms ordered from highest to lowest degree.

  List Representation: Storing only coefficients.

```python
coeffs = [3, 2, -5, 7]  # 3x³, 2x², -5x, 7
```

Methods for Polynomial Evaluation

1. Naïve Method: Simple Polynomial Evaluation

- Compute each term separately and sum them up.
- **Rewriting Example: $3x^3+2x^2-5x+7$**

Time Complexity: ***O(n^2)*** due to repeated exponentiation.

Example:

- Given Polynomial: $7 - 5x + 2x^2 + 3x^3$
- Step-by-step Calculation for `x = 2`:
  1. $7 \times 2^0 = 7$
  2. $-5 \times 2^1 = -10$
  3. $2 \times 2^2 = 8$
  4. $3 \times 2^3 = 24$
- Final Sum: $7 + (-10) + 8 + 24 = 29$
- Output: `29`

```python
def naive_eval(coeffs, x):
    result = 0
    for i in range(len(coeffs)):
        result += coeffs[i] * (x ** i)
    return result

coeffs = [7, -5, 2, 3]  # Order: x^0, x^1, x^2, x^3
print(naive_eval(coeffs, 2))  # Output: 29
```

Explanation

Steps Involved in `naive_eval(coeffs, x)`

1. **Initialize result** → Set `result = 0` to store the computed polynomial value.
2. **Loop through coefficients** → Iterate over `coeffs` using an index `i`, where each coefficient represents xix^ixi.
3. **Compute each term** → Multiply `coeffs[i]` by xix^ixi (`coeffs[i] * (x ** i)`).
4. **Accumulate result** → Add each computed term to `result`.
5. **Return final value** → After the loop, return the final `result`.

2. Horner's Method (Efficient Approach)

- Reduce computations by factoring out x.
- **Rewriting Example: $3x^3+2x^2-5x+7$**

Time Complexity: ***O(n)*** (Faster than naïve method)

Example:

Mathematical Steps

1. Rewrite the polynomial using nested factorization:

$$P(x) = ((3x + 2)x - 5)x + 7$$

2. Compute step by step:
   - $3 \times 2 + 2 = 6 + 2 = 8$
   - $8 \times 2 - 5 = 16 - 5 = 11$
   - $11 \times 2 + 7 = 22 + 7 = 29$

✅ **Final Answer:** $P(2) = 29$

```python
def horner_eval(coeffs, x):
    result = 0
    for coef in reversed(coeffs):
        result = result * x + coef
    return result

coeffs = [3, 2, -5, 7]  # 3x³, 2x², -5x, 7
print(horner_eval(coeffs, 2))  # Output: 29
```

Explanation:

- **Initialize `result` to 0.**
- **Iterate through the coefficients in reverse order** (starting from the constant term).
- **At each step, multiply `result` by `x` and add the current coefficient.**
- **Repeat until all coefficients are processed, giving the final polynomial value.**
- **Return the computed result.**

3. Using NumPy's polyval() Function (Efficient & Built-in)

- Uses **vectorized operations** internally to compute polynomial values.
- NumPy is optimized for numerical computations
- **Rewriting Example: $3x^3+2x^2-5x+7$**

Time Complexity: ***O(n)*** (Internally optimized like Horner's method).

Example:

Mathematical Steps

1. Define coefficients:

$$[3, 2, -5, 7]$$

2. Use **np.polyval()** to evaluate:

$$P(2) = 3(2^3) + 2(2^2) - 5(2) + 7$$

3. Compute:

$$3(8) + 2(4) - 5(2) + 7 = 29$$

✅ **Final Answer:** $P(2) = 29$

```python
import numpy as np

coeffs = [3, 2, -5, 7]  # Represents 3x³ + 2x² - 5x + 7
x = 2
print(np.polyval(coeffs, x))  # Output: 29
```

Explanation:

- Import NumPy.
- Define polynomial coefficients: `[3, 2, -5, 7]` for **$3x^3+2x^2-5x+7$**
- Set x=2
- Use `np.polyval(coeffs, x)` to evaluate the polynomial → Output: `29`.
- Call `horner_eval(coeffs, 2)` (if defined) → Output: `29`.

Applications of Polynomial Evaluation

1. **Computer Graphics:** Bezier curves in animation.
2. **Machine Learning:** Polynomial Regression.
3. **Physics & Engineering:** Motion equations

Summary & Conclusion

- Polynomial evaluation helps compute the value of a polynomial at a given x.
- **Naïve method** is simple but inefficient.
- **Horner's method** is **fast and computationally efficient**.
- Used in **scientific computing, graphics, and AI**.

Now You Try!

Write Python code to evaluate:

$$P(x) = 2x^4 - 3x^3 + 4x^2 - 5x + 6$$

at **x = 3** using **Horner's Method**.

Thank You